IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re Application of:
Broussard

Serial No. 09/870,613

Filed: May 31, 2001

For: SYSTEM AND METHOD FOR REDUCING MEMORY USE ASSOCIATED WITH THE GRAPHICAL REPRESENTATION OF A LIST CONTROL

§ Group Art Unit: 2173
§ Examiner: Bonshock, D.
§
§ Atty. Dkt. No. AUS920010261US1
§ (5468-07400)

I hereby certify that this correspondence is being transmitted via facsimile or deposited with the U.S. Postal Service with sufficient postage as First Class Mail in an envelope addressed to: Commissioner for Patents, Washington, D.C. 20231, on the date indicated below:

09/07/2005
Date

Kevin L. Daffer

APPEAL BRIEF

Box AF
Commissioner for Patents
Washington D.C. 20231

Sir/Madam:

Further to the Notice of Appeal faxed to and received in the U.S. Patent and Trademark Office on February 7, 2005, Appellant presents this Appeal Brief. The Notice of Appeal was filed following mailing of a final Office Action on November 16, 2004. Appellant hereby appeals to the Board of Patent Appeals and Interferences from a final rejection of claims 1-4, 8, 10-12, 14, and 17-20 in the final Office Action, and respectfully requests that this appeal be considered by the Board.

I. REAL PARTY IN INTEREST

The subject application is owned by International Business Machines Corporation, a corporation having its principal place of business at New Orchard Road, Armonk, New York, 10504, as evidenced by the assignment recorded at Reel 011888, Frame 0891.

II. RELATED APPEALS AND INTERFERENCES

Notices of Appeal have been filed for the following applications, which share a common specification with the application currently on appeal.

09/870,614: Notice of Appeal filed 10/26/04; Appeal Brief filed on or about December 20, 2004.
09/870,615: Notice of Appeal filed 9/14/04; Appeal Brief filed on or about November 9, 2004.
09/870,620: Notice of Appeal filed 12/7/04; Appeal Brief filed on or about February 7, 2005.
09/870,621: Notice of Appeal filed 9/24/04; Appeal Brief filed on or about November 23, 2004.
09/870,622: Notice of Appeal filed 8/24/04; Appeal Brief filed on or about October 25, 2004.

Application serial number 09/870,621 shares similar cited art references with the present application; however, dissimilar art is cited in the present application and application serial numbers 09/870,614, 09/870,615, 09/870,620, and 09/870,622. No other appeals or interferences are known which would directly affect or be directly affected by or have a bearing on the Board's decision in this appeal.

III. STATUS OF CLAIMS

Claims 1-17 were originally filed in the present application. Claims 1, 2, 8, 10, 12, 14, and 17 were amended, claims 5-7, 9, 13, 15, and 16 were canceled, and claims 18-20 were added in the Response to Office Action Mailed March 15, 2004. Claims 1-4, 8, 10-12, 14, and 17-20 stand finally rejected under 35 U.S.C. § 103, which are the subject of this appeal. A copy of claims 1-4, 8, 10-12, 14, and 17-20 as on appeal are included in the Appendix hereto.

IV. STATUS OF AMENDMENTS

No amendments to the claims were filed subsequent to their final rejection. The Appendix hereto therefore reflects the current state of the claims.

V. SUMMARY OF THE INVENTION

Appellant's claimed invention relates to a display system (10, Fig. 1), computer-readable storage device (e.g., memory 18) and method for generating a display image. More specifically, the claimed invention relates to a system and method for reducing memory use associated with the graphical

representation of a listbox or choice control in a graphical user interface. *See*, e.g., Specification, page 30, line 4 to page 31, line 22 and the Title.

In some embodiments, the presently claimed display system may include a display (16, Fig. 1), a memory (18), and a platform-independent application program (APP 28 of Figs. 1-2) containing invocations of platform-dependent display routines to create a display image (26) when operating on a processor (12, Fig. 1) having an operating system (OS 40, Figs. 1-2). In one example, the platform-independent application program may be written in the Java programming language, and thus, may include invocations (e.g., call routines) of platform-dependent display routines (i.e., instructions containing native code) to create a display image, whose "look and feel" is dependent on the particular operating system running the Java application program. In some cases, the image created on the display may be of a listbox (140, Fig. 11) or choice (142) control.

To reduce memory use associated with the graphical representation of a listbox or choice control, the presently claimed display system may further include a platform-independent peer component (156, Fig. 12) coupled between the platform-independent application program (e.g., APP 28 of Figs. 1-2 containing list component 144 of Fig. 12) and a platform-independent software component (158, Fig. 12). In general, the platform-independent peer component (156) may be used for intercepting the invocations of the platform-dependent display routines and for routing the intercepted invocations to the platform-independent software component (158). The platform-independent software component (158) may then be used for fetching list data from memory (146) and for producing a display image of the list data upon the display without invoking the platform-dependent display routines.

In most cases, the platform-independent application program may create the list data stored in memory. More specifically, the list data may be created only once in memory (by the application program), and may not be duplicated for the operating system. In other words, no redundant copies of the list data are created for the operating system. The platform-independent software component simply fetches the original list data from memory and produces a display image of the list data upon the display without invoking the platform-dependent display routines of the platform-independent application program.

In some embodiments, the presently claimed method for generating a display image may include running a platform-independent application program (APP 28 of Figs. 1-2 containing list component 144 of Fig. 12) upon a computer (10, Fig. 1) operating under an operating system (OS 40, Figs. 1-2). In some

cases, the platform-independent application program may be used to invoke platform-dependent image display software for generating a display image (26, Fig. 2) of list data, which is created and stored in memory (146, Fig. 12) by the application program. To reduce memory use associated with the display image, however, a first platform-independent software component (156) may be used, in a preferred embodiment, for intercepting the platform-dependent invocations and routing the intercepted invocations to a second platform-independent software component (158). The second software component may be used for generating the display image without creating a copy of the list data originally stored in memory by the application program.

In some embodiments, the presently claimed computer-readable storage device (146, Fig. 12) may include: a collection of listing data containing a list of items which can be selected by a user via a pointing device (14, Fig. 1); an operating system (OS 40, Figs. 1-2) for operating a computer (10, Fig. 1) that includes the storage device; and a platform-independent application program (APP 28, Figs. 1-2) adapted to create a display image (26) of said listing data using platform-dependent display call instructions. In a particular aspect of the invention, the presently claimed computer-readable storage device may also include a first platform-independent software component (156, Fig. 12) that intercepts and transfers the platform-dependent display instructions to a second software component (158) that draws from a library of commands (160), which are independent of the operating system, for producing the display image.

VI. ISSUES

1. Whether claims 1-4, 8, 10-12, 14 and 17-20 are unpatentable under 35 U.S.C. § 103(a) over U.S. Patent No. 6,727,918 to Nason et al. (hereinafter "Nason") in view of U.S. Patent No. 5,327,529 to Fults et al. (hereinafter "Fults").

VII. GROUPING OF CLAIMS

Claims 1-4, 8, 10, 11 and 17-20 (Group I) stand or fall together.
Claims 12 and 14 (Group II) stand or fall together.

The reasons why the two groups of claims are believed to be separately patentable are explained below in the appropriate parts of the Argument.

VIII. ARGUMENT

Swing was developed as part of the Java Foundation Classes in an effort to overcome the platform-dependency of AWT-based (i.e., Java) application programs. An application program interface (API) written using Swing contains no native code (i.e., instructions specific to a particular OS), and therefore, can be run on substantially any OS without changing the look and feel of the application. As such, Swing may be used to overcome some of the problems and shortcomings that are inherent to AWT-based application programs. *See*, e.g., Specification, page 9, lines 14-30; page 19, line 25 to page 20, line 4; page 30, lines 4-6.

For example, the AWT implementation of a listbox or choice control creates redundant copies of the list data stored in memory, and thus, may incur substantial memory overhead. For example, when a listbox (140, Fig. 11) or choice (142) control is created by an AWT-based application program, the application program stores the list data associated with the control in a memory array (146). When the listbox or choice control is to be displayed, the AWT-based application program creates a heavy-weight peer component (148) for implementing the target list component (144). Because the peer component is heavy-weight (i.e., contains native code specific to a particular OS), the peer component implements the target list by referencing a corresponding object in the operating system (OS 152). To do this, the peer component executes a copy loop (150), in which all of the items in the original memory array (146) are duplicated in a second memory array (154) for access by the operating system. Though redundant memory allocation may not be a serious problem for short lists, significant time and system resources may be wasted if the list is relatively long. *See*, e.g., Specification, page 30, lines 6-20 and Fig. 11.

Therefore, a need exists for an improved system and method for displaying a listbox or choice control in a graphical user interface. In a preferred embodiment, the improved system and method would reduce memory use associated with the graphical representation of a listbox or choice control by preventing the creation of redundant copies of list data. *See*, e.g., Specification, page 30, lines 20-22.

The invention as recited in claims 1-4, 8, 10-12, 14, and 17-20 addresses the above-mentioned need by providing a display system, computer-readable storage device and method for generating a display image of a listbox or choice box control. In general, the presently claimed case provides a system of software components – referred to as "AWTSwing" components – that enable a Swing listbox or choice control to be indirectly substituted for an AWT listbox or choice control. However, direct

substitution of Swing and AWT components presents many problems, and as such, is often avoided by conventional programmers. *See*, e.g., Specification, page 22, lines 1-28.

When an AWT listbox or choice control (144, Fig. 12) is created by an application program (e.g., a Java application), the list data associated with the control is stored in memory (146), similar to the AWT implementation, and a peer component (156) is created for displaying the control. However, unlike the AWT implementation, the peer component created by the present invention is a lightweight (i.e., platform-independent) "AWTSwing" peer component. The lightweight peer component is used to implement the control, not as a duplicate list obtained from the operating system, but as a lightweight Swing component (160). In other words, the lightweight peer component (156) is created for intercepting and routing call routines from an AWT listbox or choice control (144) to a behavioral model (158) associated with a corresponding Swing control (e.g., Swing JList 160). This is beneficial because, unlike AWT listbox/choice controls, Swing listbox/choice controls do not create a redundant copy of the list originally stored in memory, thereby saving both time and memory space. *See*, e.g., Specification, page 30, line 24 to page 31, line 15; and Fig. 12.

As described in more detail below, none of the cited art, either separately or in combination, provides teaching, suggestion or motivation for the presently claimed display system, computer-readable storage device or method for generating a display image of a listbox or choice control. More specifically, the cited art fails to disclose (among other things) a platform-independent peer component for intercepting invocations (i.e., call routines) of platform-dependent display routines and for routing the intercepted invocations to a platform-independent software component, which is configured for producing a display image of list data without invoking a platform-dependent display routine. Thus, the teachings of the cited art cannot be used to render the limitations of the presently claimed case unpatentable.

ISSUE 1 ARGUMENTS

Patentability of Group I Claims 1-4, 8, 10, 11, and 17-20:

1. **Nason fails to provide teaching or suggestion for a display system including: (i) a platform-independent software component for fetching list data from memory and for producing a display image of the list data without invoking platform-dependent display routines, and (ii) a platform-independent peer component for intercepting invocations of the platform-dependent display routines from a platform-**

independent application program and for routing the intercepted invocations to the platform-independent software component.

Independent claim 1 states in part:

> A display system, comprising ... platform-independent application program containing invocations of platform-dependent display routines ... a platform-independent software component for fetching list data from the memory for producing a display image of the list data upon the display without invoking a platform-dependent display routine; and a platform-independent peer component coupled between the platform-independent software component and the platform-independent application program, for intercepting said invocation of platform-dependent display routines and for routing the intercepted invocations to said platform-independent software component.

Independent claim 17 (a computer readable storage device) recites a similar limitation. Therefore, claims 1 and 17 provide a display system and storage device, in which a platform-independent peer component (e.g., AWTSwing List Peer 156, Fig. 12) is coupled between a platform-independent application program (e.g., a Java application program) and a platform-independent software component (e.g., AWTSwing List Model 158). The platform-independent peer component is created for intercepting platform-dependent invocations (e.g., call routines) of the application program and for routing the intercepted invocations to the platform-independent software component. In the embodiment of claim 1, the platform-independent software component may be used for fetching list data from memory (146) and for producing a display image of the list data without invoking platform-dependent display routines. In the embodiment of claim 17, the platform-independent software component may produce the display image of the list data by drawing from a library of commands (e.g., within Swing JList 160) that are independent of the operating system.

Nason discloses "a technique for controlling allocation and content of display space among one or more user interfaces, operating systems or applications permitting an application or parallel graphical user interface (GUI) to operate outside the desktop, the area designated for display of the native operating system interface and it's associated applications." (Nason, column 2, lines 40-46). Nason, however, fails to provide teaching or suggestion for the presently claimed display system, which specifically includes: (i) a platform-independent software component for fetching list data from memory and for producing a display image of the list data without invoking platform-dependent display routines, and (ii) a platform-independent peer component for intercepting invocations of the platform-dependent display routines from a platform-independent application program and for routing the intercepted invocations to the platform-independent software component.

With regard to the presently claimed peer component, the Examiner suggests that "Nason teaches a peer component for dealing with two different interfaces", but admits that Nason "doesn't specifically disclose a peer component that routes the invocations between software components, or the elements being specifically list related." (Final Office Action, page 3). The Appellants appreciate the Examiner's recognition of the lack of teaching within Nason for the presently claimed peer component, as recited in claims 1 and 17. In addition, Appellants contend that Nason also lacks teaching or suggestion for claim elements, other than those specifically mentioned in the Office Action.

In addition to the presently claimed peer component, Nason fails to provide teaching or suggestion for the platform-independent software component recited in present claims 1 and 17. For example, statements in the Final Office Action admit that Nason fails to provide teaching or suggestion for "the elements being specifically list related" (Final Office Action, page 3). For at least this reason, Nason cannot be relied upon to provide teaching or suggestion for a software component that fetches list data from memory for producing a display image of the list data, as recited in present claims 1 and 17. Accordingly, Nason fails to teach or suggest all limitations of present claims 1 and 17.

2. Fults cannot be combined with Nason to provide teaching or suggestion the presently claimed peer or software components.

As noted above, statements in the Final Office Action admit that Nason fails to disclose "a peer component that routes the invocations between software components, or the elements being specifically list related." (Final Office Action, page 3). However, further statements in the Final Office Action suggest that "Fults teaches a system of using two different user interfaces... similar to that of Nason, but further teaches a Generic User Interface Object Library and Controller [GUIOLC] and Specific User Interface Interpreters [SUII] that" serve to map Input/Output (I/O) requirements of a Application "to the Specific User Interface [controllers under] which the application is to be presented to the user... (see column 24, line 38 through column 25, line 10 and figure 21), and further teaches the implementation of the system using lists (see column 5, lines 35-51)." (*see*, Final Office Action, page 3 and cited passages of Fults). As such, the Examiner appears to suggest that the GUIOLC and SUII components of Fults are somehow equivalent to the presently claimed peer component, and that the mere mention of lists provides teaching or suggestion for the presently claimed software component. The Appellants respectfully disagree, for at least the reasons set forth in more detail below.

As noted in a previous Response to the Office Action mailed March 15, 2004, Fults discloses a process for designing application program user interfaces, where the process involves "creat[ing] a generic user interface description (in the form of objects with attributes and hints)" (Fults, column 15, lines 62-67). A system implementing the process of Fults "reads the [generic] description, interprets it, and produces a realization of the program user interface which visually and behaviorally conforms to the explicit and implicit guidelines of a particular style guide." (Fults, column 16, lines 10-15). To display various "gadgets" or components (e.g., windows, menus, buttons, scrolling lists, etc.) within the user interface, Fults specifically teaches that "the operating system software ... handles the details of actually selecting, arranging and otherwise managing the gadgets used to represent the component." (Fults, column 25, lines 6-10, emphasis added).

Therefore, Fults discloses a "generic" or platform-independent user interface, which when run by a particular operating system, invokes a platform-dependent display routine from the operating system to produce the images of the gadgets (much like AWT). This is contradictory to the presently claimed case, which specifically includes a platform-independent software component for fetching list data from memory and for producing a display image of the list data without invoking a platform-dependent display routine. In addition to failing to provide teaching or suggestion for the presently claimed platform-independent software component, arguments are provided below for the lack of teaching within Fults for the presently claimed peer component.

In the passage cited by the Examiner, Fults discloses that "the GUIOLC provides a series of generic UI object classes ... [that] act as in interface between the Application and the portion of the operating system software that controls the representation of a specific user interface for the Application." (Fults, column 24, line 63-68). Though Fults suggests that generic UI object classes may provide an "interface" between an application and the operating system software, Fults does not teach or suggest that the generic UI object classes (presumably, the alleged "peer component") may be used for intercepting invocations of platform-dependent display routines and for routing the intercepted invocations to a platform-independent software component, as recited in present claims 1 and 17. Instead, any invocations (e.g., I/O requirements) from the Application that may (or may not) be intercepted by the generic UI object classes would be routed to the operating system software, since Fults specifically states that "the operating system software ... handles the details of actually selecting, arranging and otherwise managing the gadgets used to represent the component." (Fults, column 25, lines 6-10, emphasis added). In addition to the passage cited by the Examiner, Appellants assert that no

teaching or suggestion can be found within other portions of Fults for the presently claimed peer component.

For at least the reasons set forth above, Nason and Fults each fail to provide teaching or suggestion for the presently claimed peer and software components. Therefore, even if Nason and Fults were combined (without sufficient motivation to do so) the combined teachings of the cited art would still fail to provide teaching or suggestion for the peer and software components, as presently claimed.

Furthermore, the cited cannot be modified to teach or suggest the presently claimed peer and software components, since Nason and Fults each fail to suggest a desirability for doing so. The mere fact that references can be combined or modified does not render the resultant combination obvious unless the prior art also suggests the desirability of the combination. *In re Mills*, 916 F.2d 680, 16 USPQ2d 1430 (Fed. Cir. 1990); MPEP 2143.01.

3. The Examiner has failed to adequately support and/or establish a *prima facie* ground of obviousness.

To establish a prima facie case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all claim limitations. MPEP § 2143. None of these three criteria have been met by the Examiner in the present case. First of all, no suggestion or motivation to modify the teachings of Nason and Fults can be found within the cited art to teach or suggest the aforementioned limitations of claims 1 and 17, as explained above in Argument 2. The criterion of a reasonable expectation of success cannot be met if no teaching, suggestion or motivation exists, because there is then nothing at which to be successful. Finally, Nason and Fults each fail to teach, and therefore, cannot be combined to teach, all of the limitations of claims 1 and 17, as explained above in Arguments 1 and 2. The third criterion recited above has therefore also not been met, and a *prima facie* case of obviousness has not been established.

Conclusion

As explained in Arguments 1-3 above, at least some limitations of claims 1 and 17, and therefore, at least some limitations of claims 2-4, 8, 10, 11 and 18-20, are not taught or suggested by the cited art. Furthermore, there is no teaching, suggestion or motivation to modify the cited art to teach the limitations of these claims. For at least the reasons set forth above, claims 1-4, 8, 10, 11 and 17-20 are patentably distinct over the cited art. Therefore, the rejection of Group I claims 1-4, 8, 10, 11 and 17-20 under 35 U.S.C. §103(a) is asserted to be erroneous.

Patentability of Group II Claims 12 and 14:

1. **Nason and Fults each fail to disclose a method for generating a display image, where the method includes intercepting platform-dependent invocations by a first platform-independent software component, and generating a display image using a second platform-independent software component without creating a copy of list data stored by an application program.**

Independent claim 12 recites: in part:

> A method for generating a display image, comprising: running a platform-independent application program ... said platform-independent application program invoking platform-dependent image display software for generating a display image; intercepting said platform-dependent invocations by a first platform-independent software component; and generating a display image using a second platform-independent software in response to said first platform-independent software component, wherein said second platform-independent software component generates a display list image without creating a copy of list data stored by said application program.

With regard to claim 12, the Examiner admits that Nason "doesn't specifically disclose a peer component that routes the invocations between software components, or the elements being specifically list related," but suggests that Fults may be combined with Nason to overcome the deficiencies therein (*see*, Final Office Action, pages 4-5). In addition, the Examiner appears to suggest that since Nason mentions the use of JAVA, the combined teachings of Nason and Fults would somehow be sufficient to render the limitations of claim 12 unpatentable (*see*, Final Office Action, pages 4-5). The Appellants respectfully disagree, for at least the reasons set forth in more detail below.

First of all, and as noted above in the Group I arguments for the patentability of claims 1 and 17, Nason and Fults each fail to teach, suggest or provide motivation for a platform-independent peer (or software) component for intercepting and routing platform-dependent invocations to a (second) platform-independent software component. The cited art also fails to teach, suggest or provide motivation for a platform-independent software component, which either fetches list data from memory for producing a display image of the list data without invoking a platform-dependent display routine (as recited in claim 1), or draws from a library of commands that are independent of an operating system for producing the display image of the list data (as recited in claim 17). In addition, the cited art cannot be combined in a manner that teaches or suggests such limitations.

For at least these reasons, the cited art cannot be relied upon for teaching the presently claimed method steps of "intercepting said platform-dependent invocations by a first platform-independent software component; and generating a display image using a second platform-independent software in response to said first platform-independent software component." As a consequence, the cited art cannot be relied upon to provide teaching for all limitations of present claim 12. As described in more detail below, the cited art also fails to disclose the additional limitation recited in claim 12, which states that "the second platform-independent software component generates a display list image without creating a copy of list data stored by said application program."

Nason and Fults each fail to provide teaching or suggestion for a platform-independent software component that generates a display list image without creating a copy of list data stored by a platform-independent application program. However, the Examiner appears to suggest that since Nason mentions the use of JAVA, such a feature may be inherently taught by Nason. For example, the Examiner suggests that Nason teaches "implementing [a] system using content and operating software in JAVA (see column 5, lines 60-63)." (Final Office Action, page 5). The Examiner also suggests that "JAVA is known in the art to comprise GUIs such as AWT, a JAVA GUI that is known to rely on the native GUI of the computer on which the application runs, and SWING, a JAVA GUI that runs uniformly on any native platform (see Microsoft Computer Dictionary pages 13 and 505)." (Final Office Action, page 5).

However, the mere mention of JAVA does not provide sufficient motivation for one skilled in the art to reasonably conclude that the system of Nason necessarily includes a first platform-independent software component for intercepting platform-dependent invocations, and a second platform-independent

software component for generating a display list image without creating a copy of list data stored by a platform-independent application program (such as a JAVA-implemented application program).

Even if one skilled in the art were to assume that the JAVA-implemented system of Nason uses either the Abstract Window Toolkit (AWT) or the Swing Toolkit for generating display images of objects, the proposed assumption would still fail to provide teaching or suggestion for the presently claimed first and second platform-independent software components. For example, if images of list data (which is not taught by Nason) were displayed using AWT components, the displayed images would be generated by creating a copy of the list data stored in memory (*see*, Specification, page 30, lines 4-20), and thus, could not be generated by the presently claimed second software component. Though redundant memory storage may be avoided if the images of list data were displayed using Swing components (which is not taught by Nason or Fults), the exclusive use of Swing components would eliminate the need for intercepting platform-dependent invocations (originating, e.g., from an AWT component), which in turn, would eliminate any need or desire for including the presently claimed first platform-independent software component. Furthermore, Nason and Fults provide absolutely no teaching, suggestion, motivation or desire for mixing Swing and AWT components, and therefore, cannot be modified to include a first platform-independent software component for intercepting platform-dependent invocations or a second platform-independent software component for receiving the intercepted invocations and generating a display list image without creating a copy of list data stored by a platform independent application program.

2. **There is no motivation to modify the teachings of Nason or Fults to provide the presently claimed method for generating a display image, where the method includes intercepting platform-dependent invocations by a first platform-independent software component, and generating a display image using a second platform-independent software component without creating a copy of list data stored by an application program.**

For at least the reasons set forth above, Nason and Fults each fail to provide teaching, suggestion, or motivation for all limitations of present claim 12, and additionally, cannot be combined to do so. In addition, Appellants wish to traverse further statements in the Final Office Action, in which the Examiner appears to rely on applicants own invention to provide motivation for modifying the combined teachings of Nason and Fults to render the presently claimed first and second platform-independent software components, or the functionality thereof, unpatentable.

With regard to dependent claim 3, which teaches that the list data is created only once in memory, the Examiner suggests that Nason teaches "the system being implemented with JAVA, which is known [to] use the SWING API, which is disclosed in the specification on page 31 to obviate the need for a redundant memory array." The Appellants disagree with the Examiner's attempt to find motivation within the applicant's own disclosure for modifying the teachings of Nason to provide teaching or suggestion for creating only one copy of the list data in memory. As noted in MPEP 2142, for example, the teaching or suggestion to make a proposed combination or modification must be found in the prior art, and not based on applications disclosure.

Unlike the presently claimed case, Nason and Fults fail to disclose or even mention the Abstract Window Toolkit (AWT), the Swing toolkit, or the limitations encountered when using either AWT or Swing components within a graphical user interface. In addition, there is absolutely no mention with Nason or Fults for the problems that may be caused by storing redundant copies of list data (due to the use of AWT list and choice box components), or any solutions that may be used to overcome such problems. Though the alternate display content controller of Nason may "include content and operating software such as Java" (Nason, column 5, lines 61-63), and Fults discloses a list component may be included within a gadget toolkit (Fults, column 5, lines 36-42), Appellants assert that there is no teaching, suggestion, or motivation within either of the prior art references to use, e.g., a Swing list component (as suggested by the Examiner) to reduce memory storage of list data by creating only one copy of the list data in memory.

Obviousness can only be established by combining or modifying the teachings of the prior art to produce the claimed invention where there is some teaching, suggestion, or motivation to do so. *In re Fine*, 837 F.2d 1071, 5 USPQ2d 1596 (Fed.Cir. 1988); *In re Jones*, 958 F.2d 347, 21 USPQ2d 1941 (Fed. Cir. 1992); MPEP 2143.01. Absent any teaching, suggestion, or motivation to do so, Appellants assert that the teachings of Nason and Fults cannot be modified to provide teaching for the presently claimed method, as recited in claim 12.

3. The Examiner has failed to adequately support and/or establish a *prima facie* ground of obviousness.

To establish a prima facie case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings.

Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all claim limitations. MPEP § 2143. None of these three criteria have been met by the Examiner in the present case. First of all, no suggestion or motivation to modify the teachings of Nason and Fults can be found within the cited art references themselves to teach or suggest the aforementioned limitations of claim 12, as explained above in Argument 2. The criterion of a reasonable expectation of success cannot be met if no teaching, suggestion or motivation exists, because there is then nothing at which to be successful. Finally, Nason and Fults each fail to teach all limitations of claim 12, as explained above in Argument 1. The third criterion recited above has therefore also not been met, and a *prima facie* case of obviousness has not been established.

Conclusion

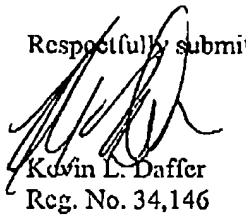
As explained in Arguments 1-3 above, at least some of the limitations recited in claim 12, and therefore, at least some limitations of claim 14, are not taught or suggested by the cited art. Furthermore, there is no teaching, suggestion or motivation within the cited art to modify the cited art to teach the limitations of claim 12. For at least the reasons set forth above, claim 12 is patentably distinct over the cited art. Therefore, the rejection of Group II claims 12 and 14 under 35 U.S.C. §103(a) is asserted to be erroneous.

IX. CONCLUSION

For the foregoing reasons, it is submitted that the Examiner's rejection of claims 1-4, 8, 10-12, 14, and 17-20 was erroneous, and reversal of the Examiner's decision is respectfully requested.

The Commissioner is hereby authorized to charge the required fee(s) to deposit account number 50-3268/5468-07400.

Respectfully submitted,



Kevin L. Daffer
Reg. No. 34,146
Attorney for Appellant

Daffer McDaniel, LLP
P.O. Box 684908
Austin, TX 78768-4908
Date: April 7, 2005
JMF

X. APPENDIX

The present claims on appeal are as follows.

1. A display system, comprising:

a display;

a platform-independent application program containing invocations of platform-dependent display routines to create a display image when operating on a processor having an operating system;

a memory;

a platform-independent software component for fetching list data from the memory for producing a display image of the list data upon the display without invoking a platform-dependent display routine; and

a platform-independent peer component coupled between the platform-independent software component and the platform-independent application program, for intercepting said invocation of platform-dependent display routines and for routing the intercepted invocations to said platform-independent software component.

2. The display system as recited in claim 1, wherein the application program creates the list data prior to fetching the list from the memory.

3. The display system as recited in claim 1, wherein the list is created only once in the memory by the application program, and not again by the operating system.

4. The display system as recited in claim 1, wherein the image of the list upon the display is created independent of the operating system.

8. The display system as recited in claim 1, wherein the platform-independent software component is part of a system of software components comprising a Java Swing application program interface (API).

10. The display system as recited in claim 1, wherein the platform-independent software component is a choice or list control.

11. The display system as recited in claim 1, wherein the application program is written in Java programming language.

12. A method for generating a display image, comprising:

running a platform-independent application program upon a computer operating under an operating system, said platform-independent application program invoking platform-dependent image display software for generating a display image;

intercepting said platform-dependent invocations by a first platform-independent software component; and

generating a display image using a second platform-independent software component in response to said first platform-independent software component, wherein said second platform-independent software component generates a display list image without creating a copy of list data stored by said application program.

14. The method as recited in claim 12, wherein said first platform-independent software component is a peer component emulating a platform-dependent peer component.

17. A computer-readable storage device, comprising:

a collection of listing data containing a list of items which can be selected by a user via a pointing device;

an operating system for operating a computer that includes the storage device; and

a platform-independent application program adapted to create a display image of said listing data using platform-dependent display call instructions;

a first platform-independent software component that intercepts said platform-dependent display instructions and transfers said display instructions to a second software component that draws from a library of commands that are independent of said operating system, for producing said display image.

18. The computer-readable storage device as recited in claim 17, wherein the platform-independent application program is written in the Java programming language.

19. The computer-readable storage device as recited in claim 18, wherein the second software component is a Java Swing component.

20. The computer-readable storage device as recited in claim 19, wherein the first software component is a peer component included within a system of software components form a AWT/Swing application program interface (API), wherein the peer component serves as an interface between the platform-dependent invocations of said application program written in Java programming language and the Swing software components.